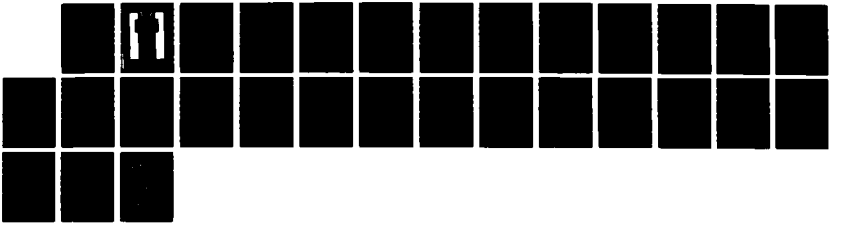
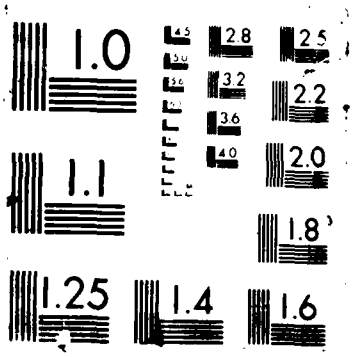
AD-A187 896 GENERATING THE STATES OF A PROBABILISTIC SYSTEM(U) 1/1
CLEMSON UNIV SC DEPT OF MATHEMATICAL SCIENCES
D R SHIER ET AL. DEC 86 TR-529 AFOSR-TR-87-1600
UNCLASSIFIED AFOSR-84-0154 F/G 12/3 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

AFOSR-TR- 87-1608

GENERATING THE STATES OF A PROBABILISTIC SYSTEM

D. R. Shier

E. J. Valvo

R. E. Jamison

DEPARTMENT
OF
MATHEMATICAL
SCIENCES

CLEMSON UNIVERSITY
Clemson, South Carolina

CLEMSON
UNIVERSITY

GENERATING THE STATES OF A PROBABILISTIC SYSTEM

D. R. Shier

E. J. Valvo

R. E. Jamison

Department of Mathematical Sciences

Clemson University

Technical Report #529

December, 1986

SECURITY CLASSIFICATION OF THIS PAGE

AD-A187 896

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 87-1608
6a. NAME OF PERFORMING ORGANIZATION / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL (If applicable) nm	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-84-0154
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete* SECURITY CLASSIFICATION OF THIS PAGE

GENERATING THE STATES OF A PROBABILISTIC SYSTEM

D. R. Shier

E. J. Valvo

R. E. Jamison

Department of Mathematical Sciences
Clemson University
Clemson SC 29634

Abstract. An important task in the evaluation of a communication or distribution system is assessing the performance of the system, when its components are subject to random failure. One approach for approximating various such performance measures is to generate a relatively small set of states of the system that covers in probability a large portion of the state space. Specifically, one would like to generate the states of the probabilistic system in order of nonincreasing probability. Rather interestingly, there is an elegant algebraic structure (a lattice) underlying this problem, and this structure can be exploited to produce a relatively effective algorithm for generating in order the states of the given system. In addition, the worst-case computational complexity of the algorithm is shown to be related to a certain algebraic invariant of the lattice.

Key words. *algorithm, lattice, graph, partial order, probability, reliability*

AMS (MOS) subject classifications: 06A10, 62N05, 68M20

1. Introduction

It is frequently of interest to evaluate the performance of a system, such as a communication or distribution system, which is composed of failure-prone components. Since the analytic calculation of most realistic performance measures (throughput, delay, reliability) is in general difficult, there is reason to investigate methods for approximating such measures. One approach for obtaining approximations involves generating a relatively small subset of the system's states that nevertheless "covers" in probability a large portion of the state space. Specifically, we consider here procedures for generating the states of the probabilistic system in order of nonincreasing probability. (As will be later seen, this can be accomplished without examining the entire state space.) Once this has been done, it is not difficult to obtain bounds on various performance measures for the system [5]. An attractive feature of this approach is that lower and upper bounds can be generated at each step, and the whole process of generating states in nonincreasing order can be continued until the bounds become sufficiently close.

Li and Silvester [5] discussed the application of this approach to several performance measures arising in the analysis of computer networks. They provided an $O(n^2k + nk \log k)$ algorithm for generating the k most probable states of a system with n components, where k must be specified in advance. An improved algorithm, that does not require k to be specified in advance, was subsequently given by Lam and Li [4]. This algorithm, with an $O(nk + k \log k)$ complexity, provides an order of magnitude improvement over the previous procedure.

Rather surprisingly, there is an elegant algebraic structure (a lattice) underlying the state space, as discussed in Section 2. This structure can in fact

be exploited to produce an algorithm for generating the most probable states of the given system, without the need to examine the entire state space. Section 3 discusses the details of this algorithm and shows how it dominates the method of [4]. In addition, the worst-case computational complexity of the algorithm is shown to be related to a certain algebraic invariant of the lattice. In Section 4, computational experience with an implementation of the algorithm is described.

2. Structure of the State Space

Consider a system with n failure-prone components 1, 2, ..., n, assumed to be independent of one another. Component i is in the operational mode with probability p_i, and in the failed mode with probability $q_i = 1 - p_i$. Moreover, it is supposed that the components have been ordered so that

$$(1) \qquad 1/2 \le p_1 \le p_2 \le \cdots \le p_n \le 1.$$

Thus, the components are ordered from least reliable to most reliable and the ratios $R_i = q_i/p_i$ are thereby placed in nonincreasing order:

$$(2) \qquad 1 \ge R_1 \ge R_2 \ge \cdots \ge R_n \ge 0.$$

Each *state* of the system corresponds to a subset X of the set of components $K = \{1, 2, \ldots, n\}$, where the elements of X represent the failed components in that state. The collection of all states, known as the *state space*, is denoted by $S = S_n$. Note that S has 2^n elements since it is the power set of K. Every state $X \in S$ has an associated probability p(X) given by

$$p(X) = \prod_{i \in \bar{X}} p_i \prod_{j \in X} q_j$$

$$= \left(\prod_{i=1}^{n} p_i\right) \prod_{j \in X} R_j$$

$$(3) \qquad = \left(\prod_{i=1}^{n} p_i\right) R(X),$$

where

$$R(X) = \prod_{j \in X} R_j.$$

Throughout, R(X) will be called the *R-value* associated with state X. The special case when $X = \phi$ (that is, when all n components are operational) will be assigned the R-value $R_0 = 1$, consistent with equation (3).

The objective here is to generate the states X in order of nonincreasing probability p(X). It will be seen that simply knowing the ordinal information conveyed by (1) provides considerable information about the probabilities of the various states. To clarify this connection, we define, for any two states $X_i, X_j \in S$, the relation $X_i \geq X_j$ if $p(X_i) \geq p(X_j)$ holds for <u>all</u> values p_i satisfying (1). It is then straightforward to show the following.

Property 1. The set S of all states forms a partially ordered set $(S, \geq)$.

We can represent the poset $(S, \geq)$ as a (directed) graph in which each state $X \in S$ corresponds to a node of the graph, labeled by the elements of the state that it represents. Namely, state $X = \{i_1, i_2, \ldots, i_k\} \in S$, where $i_1 < i_2 < \cdots < i_k$, has an associated node labeled $i_1 i_2 \cdots i_k$. The special state $X = \phi$ corresponds to the node 0.

If $X_i \geq X_j$ holds then an arc is drawn between the corresponding two nodes in the graph. Note that by equation (3) comparing $p(X_i)$ with $p(X_j)$ is equivalent to comparing $R(X_i)$ with $R(X_j)$, so that the R-values of states completely determine the arcs of the graph. To illustrate this important observation, consider the graph of the poset $(S_2, \geq)$. An arc extends from node 1 to node 2 in this graph, since $R_1 \geq R_2$. In a similar way, an arc joins node 1 to node 12 because $R_2 \leq 1$. The resulting graph of the poset $(S_2, \geq)$ is shown in Figure 1. Notice that in this particular case, the nodes are totally ordered: $0 \geq 1 \geq 2 \geq 12$, meaning that the states are ordered (from most probable to least probable) as $\phi \geq \{1\} \geq \{2\} \geq \{1,2\}$.

The graph of the poset $(S, \geq)$ can be displayed more clearly by removing all arcs that are implied by transitivity, yielding its *Hasse diagram* [2]. For

example, the poset in Figure 1 has the Hasse diagram shown in Figure 2. However, the Hasse diagram of the poset $(S_3, \geq)$, shown in Figure 3, is not a total order.

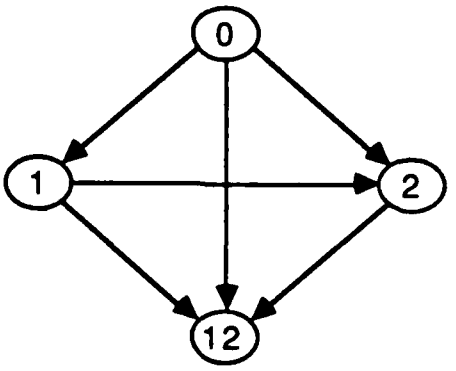


Figure 1

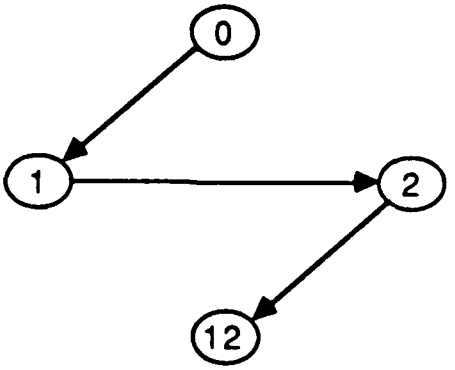


Figure 2

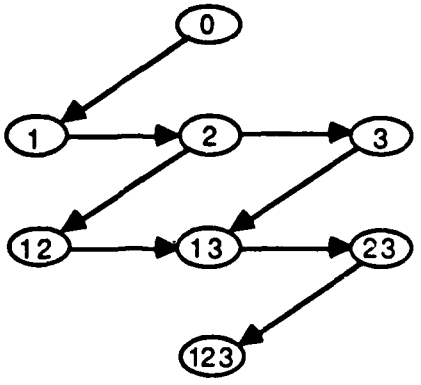


Figure 3

In general, the Hasse diagram for n components is comprised of 2^n nodes, one for each state of the system. The 2^n nodes are arranged into n+1 levels such that level j, for j = 0,...,n, contains $\binom{n}{j}$ nodes.

Each node on level j corresponds to a state with exactly j failed components. In particular, level 0 contains exactly one node (node 0), representing the state in which all components are operational, while level n contains exactly one node (node 12···n), representing the state in which all components have failed.

The arcs of the Hasse diagram can be separated into two different categories: arcs within a given level and arcs between two consecutive levels. Within level j (j = 1,...,n-1), an arc extends from node $k_1 \cdots k_{i-1} k_i \cdots k_j$ to node $k_1 \cdots k_{i-1} k_i+1 \cdots k_j$ provided k_i+1 and k_{i+1} are distinct and $k_i+1 \leq n$. Between two consecutive levels j and j+1 (j = 0,...,n-1), an arc extends from each node $k_1 k_2 \cdots k_j$ on level j, with $k_1 \neq 1$, to the node $1 k_1 k_2 \cdots k_j$ on level j+1. Figures 3 and 4 show the Hasse diagrams for the n = 3 and n = 4 cases, respectively, obtained by the above construction.

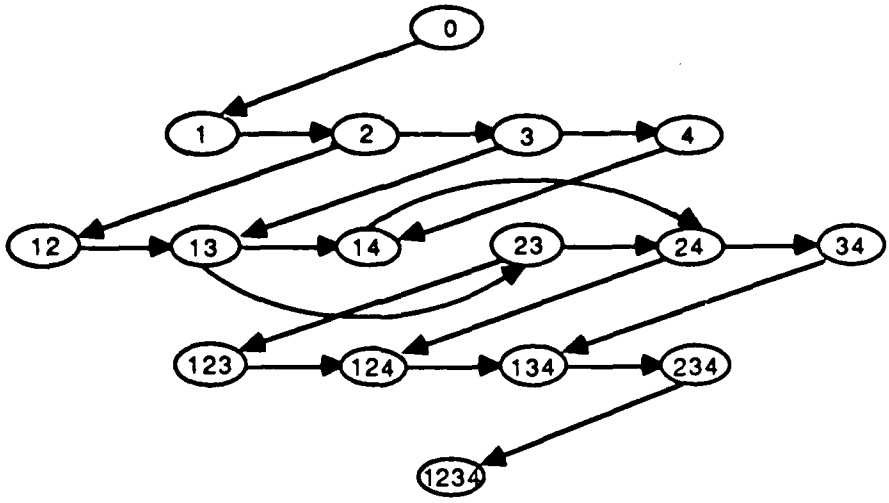


Figure 4

Property 2. *The total number of arcs in the Hasse diagram for n components is* $(n+1)2^{n-2}$.

Proof. First we count the number of arcs within each level j, for $j = 1,\ldots,n-1$. Recall that within a level there is an arc from node $k_1 \cdots k_i \cdots k_j$ to node $k_1 \cdots k_i+1 \cdots k_j$ provided k_i+1 and k_{i+1} are distinct and $k_i+1 \leq n$. We now count the number of valid nodes that admit an outgoing arc of this type. Since $k_i \leq n-1$ there are n-1 choices for k_i. The remaining j-1 integers of the label cannot equal either k_i or k_i+1. Therefore, there are only n-2 integers from which to choose the remaining j-1 integers, yielding $\binom{n-2}{j-1}$ such valid "origin" nodes with this designated k_i. Hence the number of arcs within level j is $(n-1)\binom{n-2}{j-1}$ and the total number of within-level arcs is

(4) $$\sum_{j=1}^{n-1}(n-1)\binom{n-2}{j-1}.$$

Between levels j and j+1 there is an arc from each node $k_1k_2\cdots k_j$ on level j to node $1k_1k_2\cdots k_j$ on level j+1 if $k_1 \neq 1$. Only those nodes in level j which do not contain a 1 in their label will have an arc to a node on level j+1. Since there are $\binom{n-1}{j}$ such nodes, the total number of between-level arcs is

(5) $$\sum_{j=0}^{n-1}\binom{n-1}{j}.$$

Thus, the total number of arcs M is given by

$$\begin{aligned} M &= \sum_{j=0}^{n-1}\binom{n-1}{j} + \sum_{j=1}^{n-1}(n-1)\binom{n-2}{j-1} \\ &= 2^{n-1} + (n-1)2^{n-2} \\ &= (n+1)2^{n-2}. \end{aligned}$$

Hence the total number of arcs in the Hasse diagram is $(n+1)2^{n-2}$, as claimed. ♦

An interesting feature of the Hasse diagram based on $n \geq 2$ components is that it contains two copies of the Hasse diagram on n-1 components. One of the copies is comprised of all nodes in which component n is operative. In fact this copy is an exact duplicate of the Hasse diagram for the n-1 component system. The second copy contains all nodes in which component n has failed. The labels in this copy simply have the integer n adjoined to the end of each label in the first copy. In particular, since node 0 corresponds to the empty set, node 0 in the first copy corresponds to node n in the second copy. Using this "duplication" fact and Property 2, it is easy to show the following.

Property 3. In the Hasse diagram based on n components, there are 2^{n-2} arcs joining the two copies of the Hasse diagram based on n-1 components.

The Hasse diagram of the 4 component system shown in Figure 5 illustrates the above duplication feature. The two copies of the Hasse diagram for 3 components are shown with solid lines (heavy and medium), while the arcs joining the copies are shown as dotted lines. As predicted, there are $2^{4-2} = 4$ arcs joining the two copies.

The poset $(S_n, \geq)$ has been studied in other contexts [6,7,9], and further properties are now briefly mentioned. Several of these will be useful in the subsequent discussions of Section 3. First, the poset forms a distributive lattice [9]. Essentially, it is a sublattice of the Cartesian product of n chains (totally ordered sets), and hence is distributive. Moreover, the lattice can be *ranked*: namely, it can be decomposed into subsets $P_0, P_1, ..., P_h$, such that arcs of the Hasse diagram only join nodes in consecutive sets P_k. The rank of any node is simply the sum of the integers comprising its label [7]. Thus, node 0 has rank 0, node 12···n has rank n(n+1)/2, and so the lattice has a *height* (maximum rank) h = n(n+1)/2.

In an n component system with height h, let $\mathbf{r} = (r_0, r_1, ..., r_h)$ be its *rank vector*, with $r_i = |P_i|$ signifying the number of nodes having rank i. As shown in [6,9], the rank vector is symmetric and unimodal. For example, the Hasse diagram in Figure 6 shows the rank of each node in the lattice describing a 3 component system. The height of the lattice is 6 and the rank vector is $\mathbf{r} = (1,1,1,2,1,1,1)$.

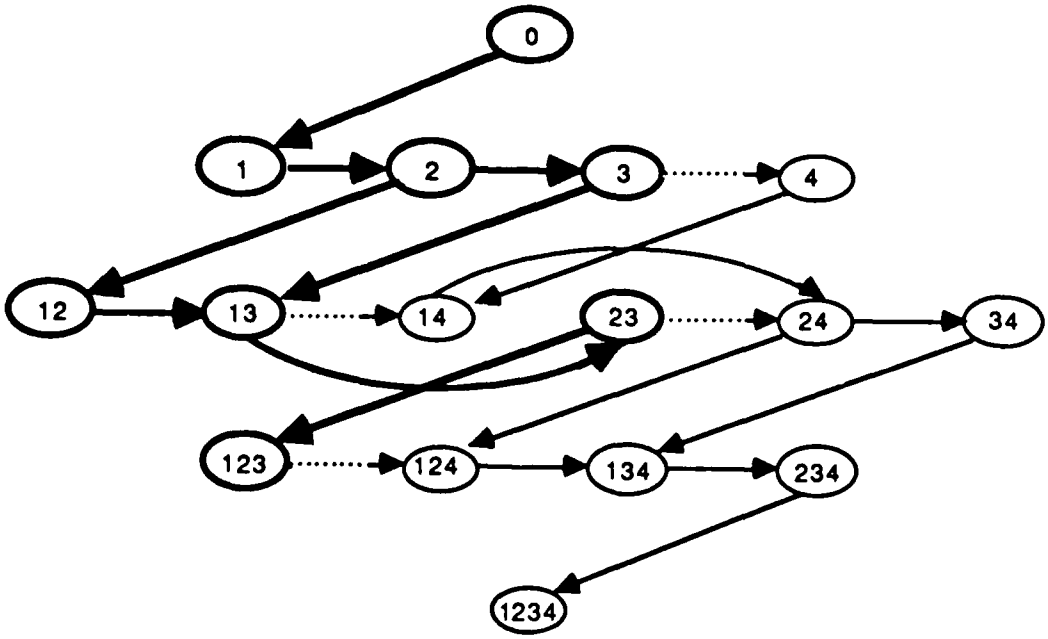


Figure 5

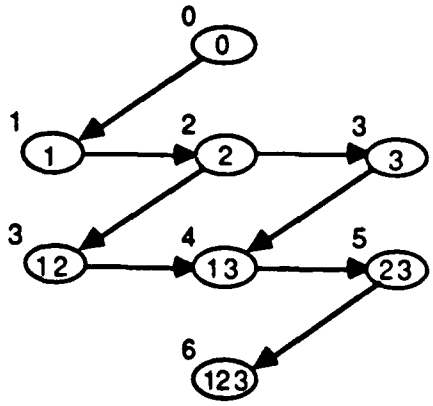


Figure 6

3. An Algorithm for State Generation

Once the state space has been identified as a poset, it is not difficult to formulate an algorithm for generating, in order, the most probable states of the system. This section describes the algorithm, which conceptually works on the Hasse diagram, and then compares it to existing procedures for solving this problem. We also determine a worst-case upper bound on the complexity of our algorithm, which is related to a certain invariant of the underlying partial order.

In the Hasse diagram for a poset, let the *indegree* of a node be the number of nodes (or *predecessors*) that cover the given node: i.e., the number of arcs entering that node in the Hasse diagram. Any (finite) poset contains at least one node with indegree 0. In our particular case, $(S_n, \geq)$ contains a single such node (namely, node 0), and it corresponds to the most probable state of the system. When this node is removed from the Hasse diagram, there will be at least one node in the reduced Hasse diagram with indegree 0. More generally, when the k most probable nodes have been removed from the Hasse diagram, there will remain some nonempty set C of nodes having indegree 0 in the reduced Hasse diagram. Any two such nodes (states) X, Y in C must be *incomparable* in the original partial order: i.e., neither $X \geq Y$ nor $Y \geq X$ holds. All such nodes are candidates for the next most probable state. Therefore, the partial information embodied in (1) is not sufficient to order the probabilities of the states represented in C; however, by comparing R-values, the next most probable state (selected from C) can be determined.

The general idea of the algorithm, then, is to at each step remove from C a node X with largest probability and then update the *candidate set* C, since the removal of X (including its arcs) may create new nodes with indegree 0. Namely, any *successors* Y of X (having X as a predecessor) will have their

indegree reduced by 1. In order to avoid looking at the entire partial order, it is useful to keep and update an *active set* A, which contains those nodes in the Hasse diagram having a predecessor that has already been removed from the Hasse diagram. A node is transferred from the set A to the candidate set C whenever its indegree decreases to 0. This process of successively removing nodes from C and updating the relevant sets can be continued until all states in S_n have been generated in order. More typically, however, the process is continued until some stopping criterion is satisfied. For example, termination may be governed by achieving some desired "coverage" of the state space. In our algorithm, termination occurs after a certain percent coverage π has been obtained, the percent coverage being the sum of the probabilities of the most probable states generated so far [5].

The procedure described above yields an algorithm to generate the most *probable states in order* of (nonincreasing) probability. The steps of algorithm GENERATE are summarized below.

Algorithm GENERATE

Input: Number of components (n), percent coverage desired (π), and component probabilities (p_1, ... , p_n) consistent with (1).

Output: States in order of nonincreasing probability until the specified percent coverage is obtained.

1. [Initialization]

   Zero := 0; Last := 12···n;

   X := Zero; C := ϕ; A := ϕ;

   Done := false; Sum := 0;

   Output: X;

2. While (not Done) and (Sum < π) do

Begin

Sum := Sum + p(X);

If X = Last then Done := true

Else Begin

2.1 Find successors of X, place them on A (if not already present) and update their indegrees;

2.2 Remove all nodes with indegree 0 from A and place them on C;

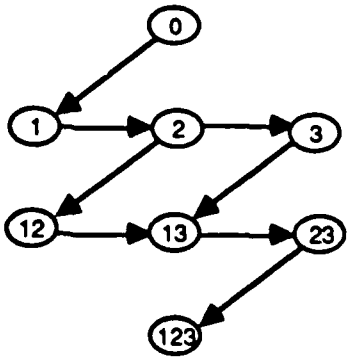
2.3 Determine and delete the node X $\in$ C having the largest R-value;

2.4 Output: X;

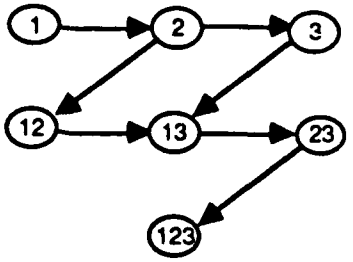
End;

End.

We illustrate the workings of algorithm GENERATE on a 3 component system with $p_1 = .7$, $p_2 = .8$, and $p_3 = .9$. Figure 7 shows the sequence of reduced Hasse diagrams encountered, as well as the status of the candidate list and the active list (before and after Step 2.2). It should be observed that there is only one place (indicated by an asterisk) in which a single arithmetic comparison is needed among the R-values to determine the next state generated. In fact, for any 3 component system, at most one comparison is ever needed to generate the states in order, showing that the simple ordering of components according to (1) provides a good deal of useful information.

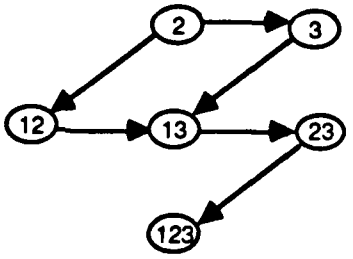


Output	Candidate	Active	Indegree
0	Φ	1	0



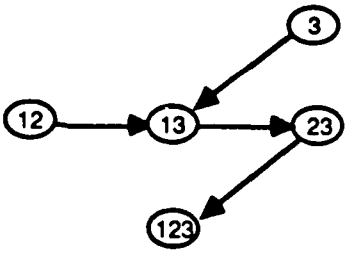
Candidate	Active
1	Φ

Output	Candidate	Active	Indegree
1	Φ	2	0



Candidate	Active
2	Φ

Output	Candidate	Active	Indegree
2	Φ	3	0
		12	0



Candidate	Active
3	Φ
12	

Output	Candidate	Active	Indegree
3	12	13	1

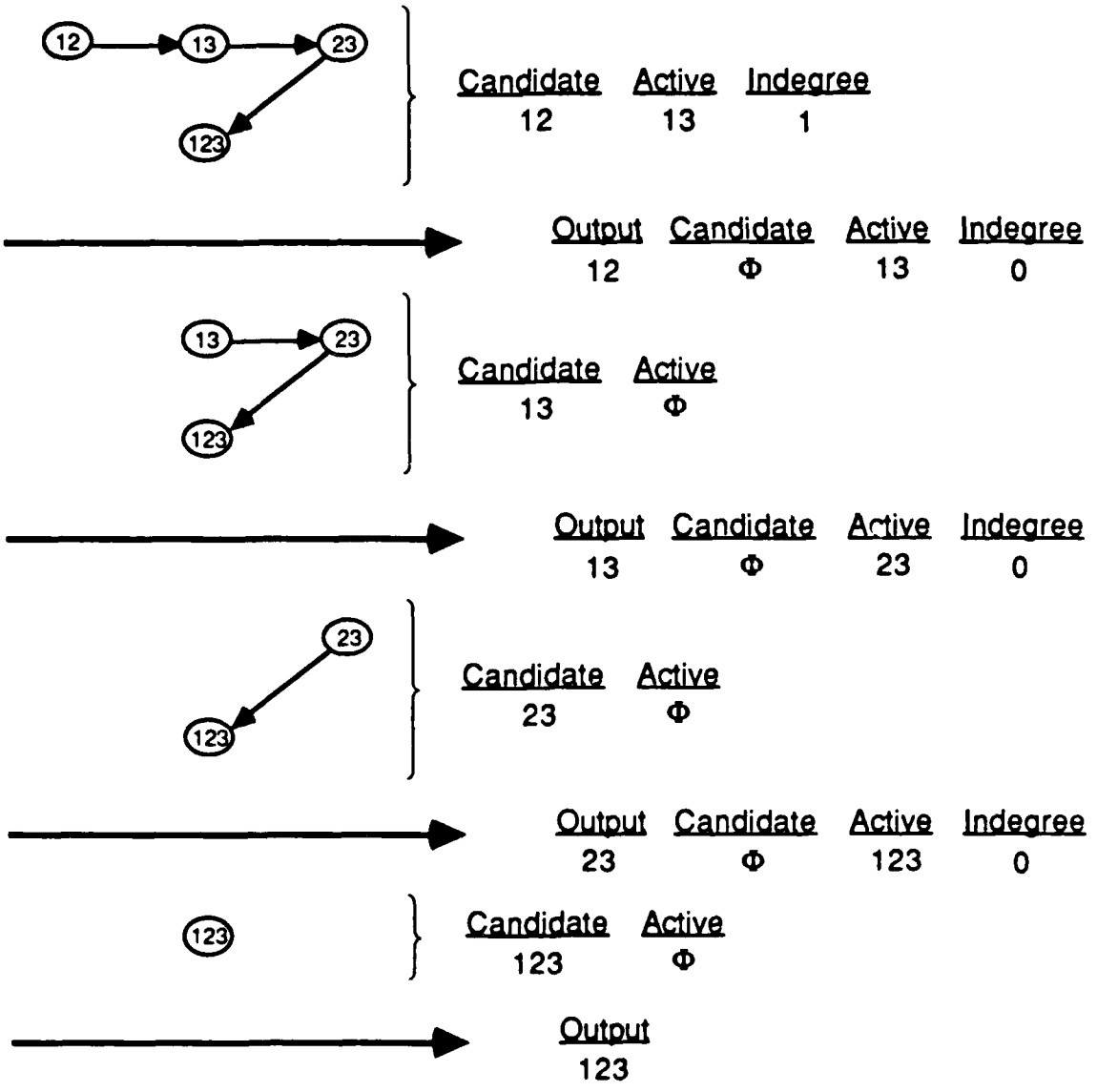


Figure 7

We now establish some properties of the sets used in algorithm GENERATE. Let D denote the set of elements deleted from the poset at some step of the algorithm; thus, the states in D have already been correctly generated in order. The *neighborhood* of D in $(S_n, \geq)$ is defined as $\Gamma(D) = \{X \notin D: X \text{ has a}$

predecessor $Y \in D$} and equals $C \cup A$. A set $K \subseteq S_n$ is termed *convex* [3] if for any $X, Z \in K$ and $Y \in S_n$ then $X \geq Y \geq Z$ implies $Y \in K$.

Lemma 1. The set D is a convex subset of S_n.

Proof. Suppose $X, Z \in D$ with $X \geq Y \geq Z$. Since Y is more probable than Z it must be removed before Z. Because $Z \in D$, it follows that $Y \in D$. ♦

Theorem 1. At ar step of the algorithm, $|\Gamma(D)| \leq |D|$.

Proof. We establish a one-to-one mapping from $\Gamma(D)$ to D. Suppose that node X is in $\Gamma(D)$. Then $X = i_1 i_2 i_3 \cdots i_m \cdots i_t$ has some predecessor $Y \in D$, say $Y = i_1 i_2 i_3 \cdots i_m\text{-}1 \cdots i_t$. (If there are several predecessors, the choice can be made arbitrarily.) Now define the map $\psi: \Gamma(D) \rightarrow D$ using $\psi(X) = i_1 i_2 i_3 \cdots i_{m-1}\, i_{m+1} \cdots i_t$. Note that $0 \geq \psi(X) \geq Y$, where node 0 is the most probable state; since $0, Y \in D$ then the convexity of D yields that $\psi(X) \in D$.

Now suppose that there is some $X' \in \Gamma(D)$, $X' \neq X$, with $\psi(X') = \psi(X)$. Then X′ must have a predecessor $Y' \in D$, which we assume has the form

$$Y' = i_1 i_2 i_3 \cdots i_{m-1}\, i_{m+1} \cdots i_{r-1}\, k\, i_r \cdots i_t.$$

(A similar argument governs other placements of the index k.) Since the indices satisfy $i_m < i_{m+1} < \ldots < i_{r-1} < k < i_r < \ldots < i_t$, it can be verified that $Y \geq X \geq Y'$ holds. Again by the convexity of D and the fact that $Y, Y' \in D$ it follows that $X \in D$, a contradiction. Thus, the map is one-to-one and the stated result follows. ♦

In order to implement algorithm GENERATE in an efficient manner, it is useful to maintain the candidate set C as a heap [8] and the active set A as a hash table [1]. In this way, selection and deletion of the most probable node

from C (Step 2.3) can be done in time logarithmic in the size of the current candidate set. Using a hash table generally allows the insertion and the location of an element in constant time. Since there can be at most O(n) successors of a given node, the remaining steps can be carried out in O(n) time per deleted node. Thus, if a total of k nodes (states) are output before termination of the algorithm occurs, then the computational complexity of algorithm GENERATE can be expressed as O(nk + k log m), where m is the maximum size of the heap occurring during execution of the algorithm.

By Theorem 1, we do know that |C| is in general less than k, and so a (weak) upper bound on m is k. As a matter of fact, the algorithm of Lam and Li [4] also uses a heap whose maximum size is k, yielding the k log k term in their complexity estimate O(nk + k log k). Thus, Theorem 1 shows that our algorithm dominates that in [4] in both computational effort and storage, since our candidate set is (in the worst case) smaller than the corresponding set used in the algorithm of Lam and Li. In order to get a sense of the difference between the sizes of these sets in practice, we summarize the results of running the two algorithms on the specific example with $p_1 = .55$, $p_2 = .6$, $p_3 = .7$, $p_4 = .8$, $p_5 = .9$ (the results obtained here are typical). Table 1 shows for this example (run until a coverage of $\pi = 0.90$ was achieved) the sizes of the respective candidate sets along with the most probable state output at each iteration. Notice that it required the generation of 16 states to obtain a 90% coverage of the state space. The candidate set for GENERATE stays much smaller than the corresponding set used in [4]. Moreover, the active list required for our algorithm required a maximum size of 4 in this example. These findings, that neither the active list nor the candidate list in GENERATE become very large, are confirmed by the empirical results presented in Section 4.

SIZE OF CANDIDATE SETS

Iteration	Lam and Li	GENERATE	Most Probable State
–	–	–	0
0	1	1	1
1	2	1	2
2	3	2	12
3	4	1	3
4	5	2	13
5	6	2	23
6	7	2	4
7	8	3	123
8	9	2	14
9	10	2	24
10	11	3	124
11	12	2	5
12	11	2	34
13	12	2	15
14	11	2	134

Table 1

The computational complexity of algorithm GENERATE, both theoretically and empirically, is determined in large part by the effort needed to process the candidate set C. While Theorem 1 provides the upper bound k = |D| on the maximum size of the set C, we now explore an alternative upper bound related to the structure of the poset $(S_n, \geq)$.

An important observation, made earlier, is that all elements in C must be incomparable in the partial order; since they all have indegree 0 in the reduced Hasse diagram, no two can lie on the same chain. In other words, the elements of C form an *antichain* [2] in the poset. Thus, an upper bound on the maximum size m of C is the size μ of the largest antichain in $(S_n, \geq)$.

Stanley [9] and Proctor [6,7] have studied in other contexts this same poset, referred to as M(n), and have shown some rather deep results concerning its structure. Specifically, as mentioned earlier, the elements of the poset can be partitioned into sets P_i of rank i. Moreover, the poset is *Sperner*, meaning that

the size of the largest antichain is the same as the maximum size of the sets P_i. Since the poset is rank unimodal and rank symmetric [6,9], a maximum-sized antichain occurs for a set P_j at the half-height δ of the lattice: namely, at rank

$$\delta = \left\lfloor \frac{h}{2} \right\rfloor = \left\lfloor \frac{n(n+1)}{4} \right\rfloor .$$

As a result, we obtain an upper bound $\mu = |P_\delta|$ on the maximum possible size of the candidate set. Since the rank of a node $i_1 i_2 \cdots i_t$ is simply the sum of its constituent integers i_j, another way of expressing this upper bound μ is as the number of partitions of δ into distinct parts, none of which exceed n. Table 2 shows the values of μ for the range n = 5, 6, ..., 25. Also shown in the table is the actual maximum size of the candidate list that occurred for typical examples having the indicated number of components and which were run until 95% coverage was obtained. While the upper bound μ grows rapidly, it appears to be a vast overestimate of the value observed in practice, especially for larger n. This suggests that the empirical behavior of algorithm GENERATE may be considerably better than its worst case behavior. In order to explore the empirical performance of the algorithm, the next section presents results obtained by executing the algorithm on a series of test problems.

n	δ	μ	Maximum Size Observed
5	7	3	3
6	10	5	3
7	14	8	5
8	18	14	5
9	22	23	5
10	27	40	9
11	33	70	11
12	39	124	14
13	45	221	20
14	52	397	28
15	60	722	39
16	68	1314	45
17	76	2410	63
18	85	4441	97
19	95	8220	133
20	105	15272	183
21	115	28460	258
22	126	53222	355
23	138	99820	408
24	150	187692	577
25	162	353743	809

Table 2

4. Computational Results

In this section, we assess the empirical computational complexity of algorithm GENERATE. In addition, these computational results indicate how the number of states needed to obtain a specified coverage π and how the size of the candidate set grow with n. A series of examples were analyzed in which the set of p_i's defining the n-1 component system is a subset of the set of p_i's for the n component system (n = 5, 6, ..., 25). All of the experimental data were obtained using an implementation of algorithm GENERATE in Pascal on the IBM 3081-K at Clemson University.

Table 3 shows, for each specified coverage level π, the number of components (n), the number of states generated (k), the maximum size of the candidate set occurring (m), and the elapsed CPU time required (t, in seconds). First, it is observed that as n increases, the number of states needed to achieve the desired coverage represents a smaller and smaller proportion of the total number of states 2^n. In particular, even for $\pi = .95$, less than 0.4% of the states are generated for n = 20 and less than 0.08% for n = 25. As a result, we should anticipate that $m << \mu$, since in order to achieve the maximum antichain size $\mu = |P_\delta|$ roughly 50% of the states need to be generated. Comparison of the values of m in Table 3 with those of μ in Table 2 corroborate this expectation.

Table 3 also shows that the size of the candidate set is typically much smaller than the number of states generated. This again confirms the superiority of algorithm GENERATE over the algorithm described in [4]. While the maximum size m of the candidate set does not get excessively large in these examples, it does get sufficiently large to require implementation as a heap to allow for the efficient insertion and deletion of elements.

	PERCENT COVERAGE								
	85%			90%			95%		
n	k	m	t	k	m	t	k	m	t
5	4	2	.002	5	2	.002	8	3	.004
6	5	2	.002	7	2	.003	11	3	.005
7	7	2	.003	11	3	.005	16	5	.008
8	9	2	.003	14	5	.006	22	5	.012
9	17	5	.007	24	5	.011	42	5	.022
10	28	5	.015	42	5	.024	74	9	.045
11	35	5	.020	55	8	.033	101	11	.065
12	57	7	.036	90	10	.059	170	14	.121
13	85	10	.055	138	13	.096	268	20	.204
14	122	14	.085	202	17	.152	405	28	.330
15	202	16	.153	336	21	.274	696	39	.611
16	208	19	.171	349	25	.305	740	45	.698
17	344	21	.308	597	37	.559	1275	63	1.301
18	542	32	.525	962	56	.993	2094	97	2.358
19	674	42	.671	1207	65	1.307	2730	133	3.222
20	916	58	.922	1671	88	1.819	3892	183	4.680
21	1324	78	1.464	2453	125	2.926	5877	258	7.962
22	2197	97	2.711	4119	169	5.492	10036	355	15.354
23	2430	109	3.060	4619	187	6.290	11518	408	18.052
24	3711	150	5.217	7200	270	10.981	18291	577	33.533
25	4956	204	6.562	9791	370	14.371	25552	809	46.420

Table 3

Finally, Table 3 shows that the computation time required in these examples is really quite modest. A multiple regression analysis of the data shown in the table was performed using the following model:

$$t = \beta_1 + \beta_2 kn + \beta_3 k \log m.$$

This model accounted for over 99.7% of the variation in the data and appeared to provide a quite satisfactory fit, thus substantiating the computational complexity model discussed in Section 3.

5. Conclusions

The objective of this paper has been to describe a fairly applied setting in which the identification of an underlying algebraic structure aids considerably in understanding the original problem. Specifically, being able to place the components in order of nondecreasing reliability provides a good deal of information about the relative probability of states in the state space. In addition, this algebraic viewpoint leads quite naturally to an algorithm for solving the original problem. It is of interest that the analysis of this derived algorithm itself is aided by studying an algebraic concept: the maximum-sized antichain in the lattice. Computational results are presented to complement the theoretical findings, and they indicate that the algorithm described here is reasonably effective in practice.

Acknowledgements

The authors are indebted to Jim Lawrence for his helpful and insightful comments. The work of the first author was supported by the United States Air Force Office of Scientific Research (AFSC) under Grant AFOSR-84-0154.

REFERENCES

[1] A. AHO, J. HOPCROFT AND J. ULLMAN, *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass., 1983.

[2] K. BOGART, *Introductory Combinatorics*, Pitman, Marshfield, Mass., 1983.

[3] G. GRÄTZER, *Lattice Theory*, W. H. Freeman, San Francisco, 1971.

[4] Y. LAM AND V. LI, An Improved Algorithm for Performance Analysis of Networks with Unreliable Components, Technical Report, Department of Electrical Engineering, University of Southern California, 1985.

[5] V. LI AND J. SILVESTER, Performance Analysis of Networks with Unreliable Components, IEEE Trans. Comm., COM-32 (1984), pp. 1105-1110.

[6] R. PROCTOR, Representations of sl(2,C) on Posets and the Sperner Property, SIAM J. Alg. Disc. Meth., 3 (1982), pp. 275-280.

[7] R. PROCTOR, Solution of Two Difficult Combinatorial Problems with Linear Algebra, Amer. Math. Monthly, 89 (1982), pp. 721-734.

[8] R. SEDGEWICK, *Algorithms*, Addison-Wesley, Reading Mass., 1983.

[9] R. STANLEY, Weyl Groups, the Hard Lefschetz Theorem, and the Sperner Property, SIAM J. Alg. Disc. Meth., 1 (1980), pp. 168-184.

END

DATE

FILMED

FEB.

1988